



Technical Documentation for the Web Solution

Version_20251128

All the documentation it is allocated in : <https://github.com/Vetworc/docs>





This project has been funded with the support from the European Commission

This publication reflects the views only of the author, and the Commission can not be responsible for any use which may be made of the information contained therein.



WMS Installation

This guide provides a detailed process for setting up, configuring, and deploying the Workshop Management System (WMS). It includes the necessary steps to ensure the system is up and running, from installing the operating system to configuring Docker and the WMS system. A discussion about hardware requirements can be found in SERVER SPECIFICATIONS.

Quick installation guides are available for SERVER PRESREQUISITES and WMS INSTALLATION.

The process consists of the following phases:

- Server installation.
- Network configuration.
- Installation of Docker support.
- Cloning the latest version of the software.
- WMS configuration parameters.
- Creation and startup of Docker containers.
- Proxy configuration.

Technical presentations

- 01 - System messagery.
- 02 - System overview.
- 03 - Software development.

Populating system with services

- 01 Guacamole User Creation https://youtu.be/dcHlo_Shi30
 - 02 Creation of WMS Resources <https://youtu.be/wuDT6mhu-jo>
- Consult also WMS SERVICES
- 03 User Management <https://youtu.be/4VhPb3sXDwU>
 - 04 SCORM Voucher <https://youtu.be/8l-BtPi2IFI>





- 05 SCORM Task

https://youtu.be/fnr5FrBVh_8

Contributing to Vetworc project

To contribute to the Vetworc project, you should review the following documents:

- CONTRIBUTING TO VETWORC PROJECT
- DEVELOPMENT AND BRANCHING STRATEGY
- Keep in mind that you should only work on the develop branch.

Issue Management

Bug and feature request management will be carried out through the GitHub platform following the process described in REPORTING BUGS.



WMS Hardware Load Assessment and Final Recommendation

Workshop Management System (WMS) - Hardware Evaluation for IT Decision-Makers

.1 Strategic Overview and Key Considerations

When selecting hardware for the Workshop Management System (WMS), it is crucial to prioritize components with the greatest impact on performance and scalability. Based on the system architecture and expected workload, we recommend the following strategy:

- CPU is the primary performance driver:

Both VNC remote desktop sessions and Flask applications are CPU-intensive. Since upgrading the CPU after deployment is complex, choosing a capable processor from the start is essential.

- Memory (RAM) and storage can be upgraded later:

The selected server allows straightforward upgrades for increased RAM or faster storage (e.g., SSDs) as usage grows. Higher disk speeds particularly benefit database transactions, improving read/write operations and overall responsiveness.

- Highly scalable architecture:

- ✓ Remote desktop services can be offloaded to a dedicated server if VNC sessions become the main bottleneck. This allows for easy expansion without affecting core WMS functions.
- ✓ Metrics collection and visualization (e.g., Elasticsearch and Grafana) can migrate to another server as system load increases.
- ✓ The core server will continue to serve essential services, ensuring longevity and investment protection.

The proposed solution is designed to meet current needs while offering a clear, cost-effective path to scale as demand grows.

2.2. System Overview and Operational Context

The Workshop Management System (WMS) is designed to facilitate remote desktop connections to workstations located in workshops or cells. Its primary function is to channel VNC connections from students to these devices, enabling seamless access to industrial software and practical environments.

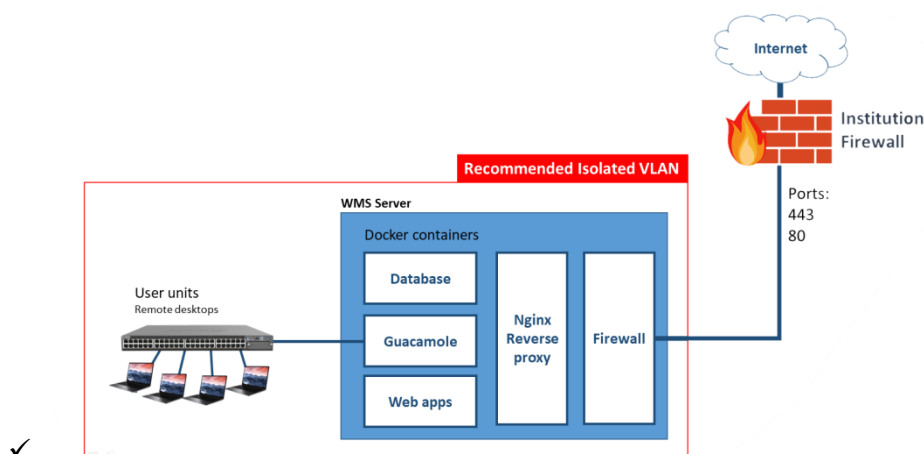
2.2.1. Deployment Environment

Network segmentation: Each workshop area (referred to as a cell) is recommended to be configured within its own VLAN or subnet, with a dedicated or shared gateway connecting to the institution's firewall or router. This improves network segmentation and enhances security.

Server location: The WMS server should be placed in a network zone accessible from the internet, ideally in a DMZ or an intermediate security area. This setup provides controlled external access while safeguarding internal network resources.

Network and firewall considerations:

- ✓ The WMS (particularly the Docker container running guacd) must connect to the VNC servers hosted on user nodes (the workstations in the cell offering remote desktops).
- ✓ Port 443 (HTTPS) must be publicly accessible, routed through a reverse proxy server that internally forwards requests to the WMS.
- ✓ Firewall and routing rules must allow necessary connections while minimizing exposure, ensuring that only essential ports and services are reachable.



2.2.2 System Responsibilities

The WMS manages the following core services:

- ✓ Remote desktop (VNC) channeling: Enables students to access workshop devices with medium graphical workloads (e.g., TIA Portal, CNC software).
- ✓ Web-based services (Flask applications): Provides platforms for session management, grading, and access to workshop tools.
- ✓ Database operations: Logs events like logins, task submissions, and session heartbeats. Faster storage solutions improve read/write speeds.
- ✓ Metrics collection: Receives regular data submissions (every 10 seconds) from up to 40 user nodes, allowing real-time and centralized monitoring.

2.2.3. Key Terminology

- ✓ VNC Session: Remote desktop connection lasting 30 minutes to 2 hours. CPU-intensive with moderate RAM usage.
- ✓ Flask Session: User interaction with WMS web services; less CPU-intensive but scales with concurrent users.
- ✓ Database Transaction: Read/write operations like metric submissions or session events; benefits from faster storage.
- ✓ Metrics Submission: Data sent periodically from workshop devices requiring efficient storage writes.

2.3. Hardware Configurations Considered

Two hardware configurations were analyzed based on projected WMS workload and scalability needs:

3.1. Minimum Configuration

CPU: Intel Core i5 or AMD Ryzen 5 (4 cores, ≥ 2.5 GHz)

RAM: 16 GB DDR4

Storage: 500 GB SATA SSD



Capacity:

Up to 15 VNC sessions and 30 Flask sessions concurrently.

Supports up to 10 TPS (transactions per second); performance may degrade under higher loads.

Assessment:

Suitable as a starting point for basic WMS functions.

Limited scalability; higher student concurrency or database size impacts performance.

Not designed for 24/7 operation; may struggle with intensive video resources.

2.3.1. Final Selected Configuration (Recommended)

CPU: Intel® Xeon® E-2456, 3.3 GHz (6 cores / 12 threads, 18 MB cache, Turbo Boost)

RAM: 32 GB DDR5 ECC (5600 MT/s)

Storage: 600 GB SAS ISE, 12 Gbps, 10k RPM (hot-plug capable)

Capacity:

Up to 50 VNC sessions and 70 Flask sessions (120 concurrent users).

Supports up to 20 TPS; switching to enterprise SSDs could double throughput.

Smooth metrics handling from 40 devices submitting data every 10 seconds.

Why this choice?

Prioritizes CPU performance for concurrent session management.

Expandable RAM and storage for future growth.

Enterprise-grade components ensure long-term reliability.

2.4. Scalability and System Architecture

The WMS has been designed for seamless scalability:

- ✓ Remote desktop workloads: Offload to a dedicated server if VNC demand increases.
- ✓ Metrics and analytics services: Initially local (Elasticsearch and Grafana), but migratable to reduce load.
- ✓ Database scalability: Faster storage improves throughput without server reconfiguration.

Key Benefits:

- ✓ CPU investment prevents future limitations.
- ✓ RAM and storage upgrades are easy and non-disruptive.
- ✓ Faster disks improve database response times under heavy workloads.

2.5. Load Expectations and Performance

Under Expected Usage:

- ✓ VNC sessions: Up to 50 concurrent users.
- ✓ Flask sessions: Up to 70 concurrent users.
- ✓ Total concurrent users: 120 without noticeable degradation.
- ✓ Metrics: Smooth data handling from 40 devices submitting every 10 seconds.
- ✓ Database throughput: Stable up to 20 TPS (expandable to 30-40 TPS with SSDs).

If Resource Usage Doubles:

- ✓ VNC sessions: 25 users without performance issues.
- ✓ Flask sessions: 35 users reliably supported.
- ✓ Database: Latency may appear beyond 20 TPS; SSD upgrades mitigate this.
- ✓ Total users: 60 concurrent users remain manageable.



Mitigation Options:

- ✓ Migrate VNC workloads to a secondary server.
- ✓ Upgrade storage to enterprise SSDs.
- ✓ Expand RAM for sustained heavy usage.

2.6. Final Thoughts and Recommendations

- ✓ The selected Intel® Xeon® E-2456 with 32 GB DDR5 ECC RAM and 600 GB SAS drive offers a balanced and scalable solution.
- ✓ CPU-first approach: Prioritizes longevity and high performance.
- ✓ Modular upgrades: Future-proof with easy RAM and storage expansions.
- ✓ Effortless scalability: Services can migrate as WMS usage evolves.
- ✓ Enhanced storage: Upgrades significantly boost database performance.



Server Network Configuration

The network configuration of the server and the DNS servers for the domain vetworc.com will enable communication between the different systems.

Network Adapter Configuration

A static IP configuration is the most common setup. In this example, the server will have the address 10.200.8.13 with a 16-bit subnet mask.

```
network:
  ethernets:
    eth0:
      addresses:
        - 10.200.8.13/16
      nameservers:
        addresses:
          - 10.100.6.1
        search: []
      routes:
        - to: default
          via: 10.200.0.1
  version: 2
```

Network Environment

The network of our center must redirect port 443 from a public IP to the public port of the reverse proxy implemented in wms-common. Specifically, according to the Docker Compose configuration, to port 443 on the host machine.

Additionally, redirecting port 80 allows for automatic certificate renewal. If it is not possible through port 80, Nginx Proxy Manager supports certificate renewal via DNS challenges.

TODO: Restrict proxy configuration. This proxy configuration makes ports 80, 81, and 443 accessible to the host machine. In production, it is advisable to restrict this configuration.

```
vetworc_npm:
  image: 'jc21/nginx-proxy-manager:latest'
  restart: always
  ports:
    - '80:80'      # HTTP port
    - '81:81'      # Web user interface
    - '443:443'    # HTTPS port
```

DNS Configuration

DNS records need to be configured so that the services offered by this server are accessible from the internet. Specifically:

Subdomain	Service	Host IP	Host Port
wms.xabec.vetworc.com	WMS API	10.200.8.13	5201
front.wms.xabec.vetworc.com	WMS Administration Page	10.200.8.13	5200

The choice of ports 5200 and 5201 is due to their use in the Docker configuration of the containers.

WMS API

This is the wms-core container, which exposes port 5000 in Docker and port 5201 on the host machine. In production, it will not be exposed on the host machine.

TODO: In a production environment, the ports will not be accessible from the host server, but only from the reverse proxy.

We will assign the subdomain: wms.xabec.vetworc.com.

/vetworc/core/wms-core/docker-compose.yaml

WMS Administration Front-End

This is a Flask application that interacts with the WMS API to manage resources, users, and system metrics.

We will assign the subdomain `front.wms.xabec.vetworc.com`, which is exposed on **port 5000**.

`/vetworc/core/wms-admin/docker-compose.yaml`

Testing

It is advisable to verify that internet traffic is being sent to our server.

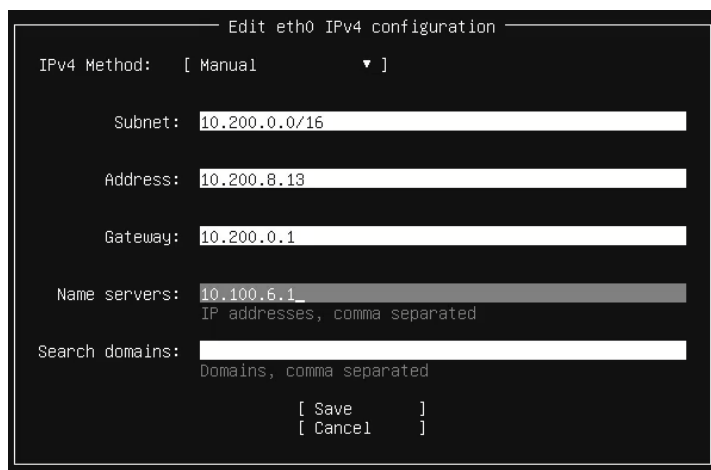
By accessing `https://wms.xabec.vetworc.com` or `https://front.wms.xabec.vetworc.com` in a browser, we should see that the traffic reaches our server.

The same should happen when accessing `http://wms.xabec.vetworc.com` or `http://front.wms.xabec.vetworc.com` on port 80.

Server Configuration

The first step is the installation of Ubuntu Server. During the installation process, keep in mind:

The server's IP configuration will depend on the instructions provided by the IT department. Please refer to the NETWORK document for more information.

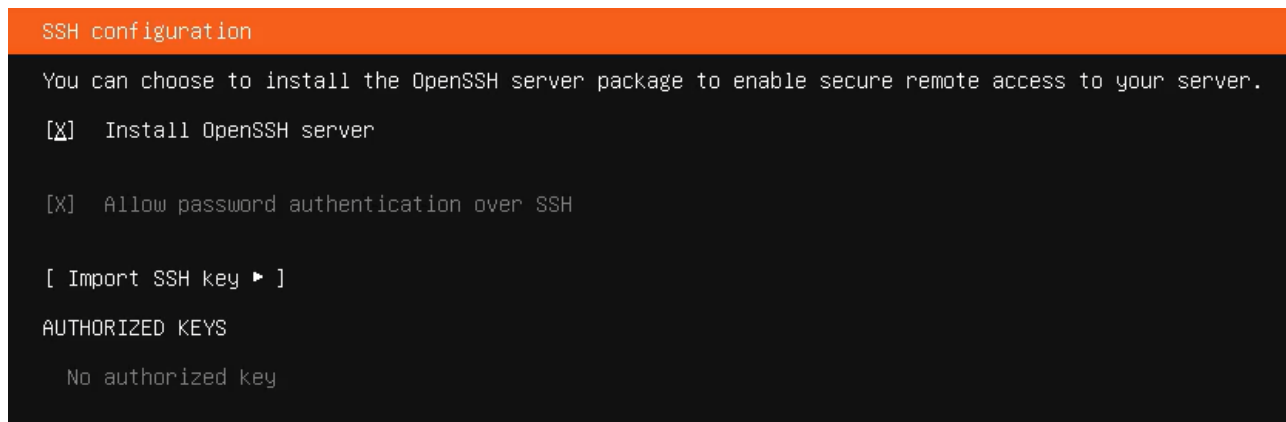


```

Edit eth0 IPv4 configuration
IPv4 Method: [ Manual ]
Subnet: 10.200.0.0/16
Address: 10.200.8.13
Gateway: 10.200.0.1
Name servers: 10.100.6.1
               IP addresses, comma separated
Search domains:
               Domains, comma separated
               [ Save ]
               [ Cancel ]

```

The SSH server must be installed



```

SSH configuration
You can choose to install the OpenSSH server package to enable secure remote access to your server.
[X] Install OpenSSH server

[X] Allow password authentication over SSH

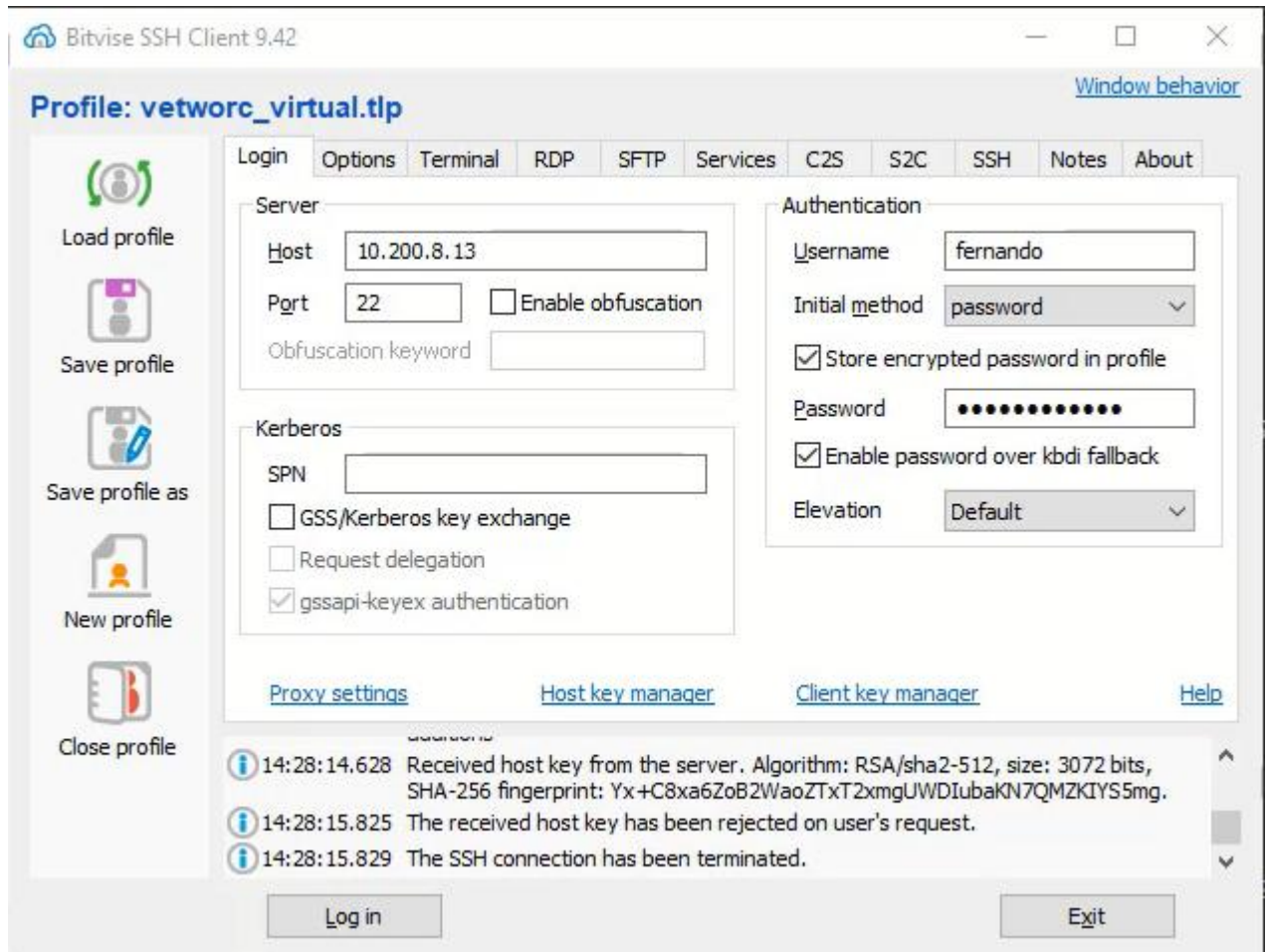
[ Import SSH key ► ]

AUTHORIZED KEYS
No authorized key

```

- Docker should not be installed, as using snap may cause issues.

As a result of the installation and configuration, we should be able to access the machine via SSH.



Docker Support Installation

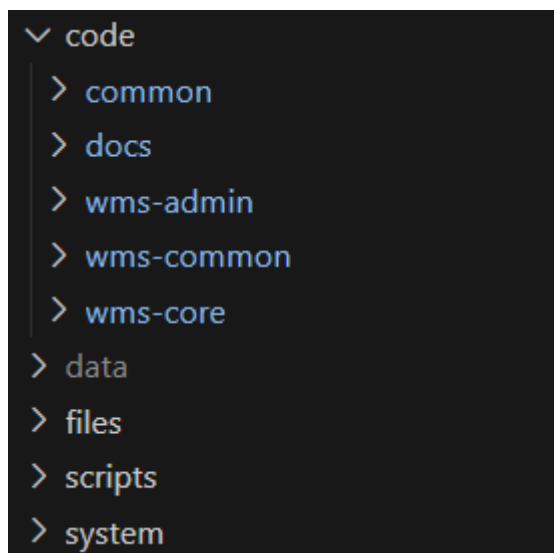
It is important not to install Docker during the Ubuntu installation process. Docker support will be installed following the instructions in the INSTALLING DOCKER document.

As a result of the process, we should be able to see the Docker service running, as well as the Docker and Docker Compose versions:

Cloning the Latest Version of the Software

In the development phase, the software will be installed by cloning this Docker repository. In this example, folder /develop will be used.

Once connected to the server, follow instructions in document REPOSITORIES. As a result there will be a structure similar to this inside the /develop/wms folder:



During the development phase, it is recommended to change the owner of the /develop folder:

```
chown -R fernando:fernando /develop
```

Configuration

The configuration is performed by editing the file /vetworc/system/config.json. The CONFIGURATION document explains the meaning of the different parameters.

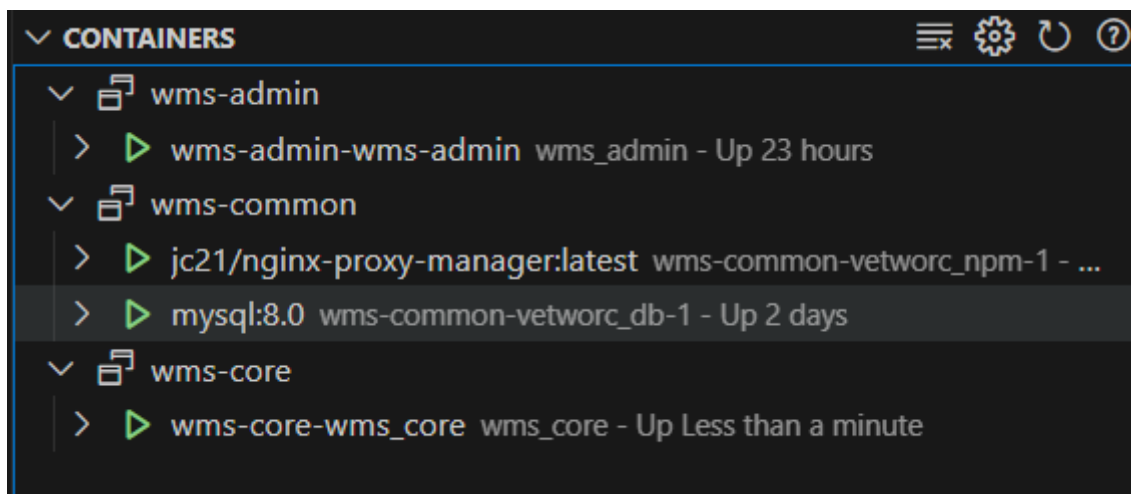
The initial configuration requires modifying the following parameters.

Parameter	Example	Description	Comments
-----------	---------	-------------	----------

Creation and Startup of Docker Containers

To create and start the Docker containers, follow the steps detailed in the DOCKER CONTAINERS document.

If the process is completed successfully, the Docker extension in Visual Studio Code will allow us to see the different containers running:



Some notes about the database are available in DATABASE.

Proxy Configuration

The proxy is a key element to make the services accessible from the internet. The NPM document details the configuration process.

Note

Depending on whether it is a development or production environment, the path could be different:

Production: /wms

Development: /develop/wms

Installing Docker

Install the necessary packages for Docker and enable the Docker repository:

```
sudo apt update
```

```
sudo apt install apt-transport-https ca-certificates curl software-properties-  
common -y
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Install Docker Community Edition:

```
sudo apt install docker-ce -y
```

Check that Docker is installed and running correctly:

```
sudo systemctl status docker
```

```
docker --version
```

Example output:

```
Docker version 27.2.1, build 9e34c9b
```

Add the user to the Docker group

To allow Docker to be used without needing sudo, add the user you use for the connection to the Docker group. For example, if your SSH connection from Visual Studio Code uses the user fernando, run:

```
sudo usermod -aG docker fernando
```

Why is this step important?

The fernando user is the one used when establishing an SSH connection to the remote system from Visual Studio Code. By adding fernando to the docker group:

- *The Docker extension in Visual Studio Code will be able to communicate directly with Docker on the remote system.*
- *You won't need to prepend every Docker command with sudo, which simplifies operations and avoids permission issues when using Visual Studio Code's integrated tools.*

After making this change, log out of the SSH session and reconnect to apply the new group membership.

Vetworc repositories

This document explains the repositories in the Vetworc project. All the necessary code is organized as a main repository (vetworc) and four repositories (vetworc submodules) that contain the documentation and different subsystems that integrate the WMS.

Repository	URL	Description	Comments
bms	https://github.com/Vetworc/bms.git	BMS repository	
cms	https://github.com/Vetworc/cms.git	CMS repository	
wms	https://github.com/Vetworc/wms.git	WMS repository	Main repository
docs	https://github.com/Vetworc/docs.git	Documentation module	
common	https://github.com/Vetworc/common.git	Module implementing common functions shared by different subsystems.	
files	https://github.com/Vetworc/files.git	Files to download...	
rdp-service	https://github.com/Vetworc/rdp-service.git	RDP service	

Cloning the Repository

- ✓ Navigate to the desired directory:

```
sudo mkdir -p /develop
```

```
sudo chown -R fernando:fernando /develop
```

```
cd /develop
```

- ✓ Clone desired repositories:

Clone the necessary repositories. In this case, we will use the wms and docs repositories.

```
git clone --branch develop --single-branch https://github.com/Vetworc/wms.git
```

Username: YOUR-USERNAME

Password: YOUR-PERSONAL-ACCESS-TOKEN

```
git clone --branch develop --single-branch https://github.com/Vetworc/docs.git
```

Username: YOUR-USERNAME

Password: YOUR-PERSONAL-ACCESS-TOKEN

Password-based authentication for Git has been removed in favor of more secure authentication methods. So, when Git asks for the password, enter your personal access token. For more information, refer to Personal Access Token Management.

You can find detailed instructions on how to generate the token in the document. GITHUB
USERS TOKEN

Setting Up the Development Environment

When setting up the development environment, once you have cloned the repository, it is very important to be sure you are working on the develop branch.

- ✓ Navigate to the Main Repository

Make sure you're in the main repository directory (the root folder of the repository):

```
cd /develop/wms
```

- ✓ Checkout the Development Branch: Make sure you are working on the develop branch:

```
git checkout develop
```

Additional Notes:

- **Why Contributions Should Not Use main:** The main branch is typically reserved for production-ready code. All contributions should be made through the develop branch, which is used for ongoing development and testing before merging into main.
- **Verify Your Git Configuration:** Before you start working, ensure that your Git configuration is set up with your name and email. This is important for proper commit history.

```
git config --global user.name "Your Name"
```

```
git config --global user.email "youremail@example.com"
```

- **Possible Common Issues:**

If a submodule doesn't update correctly, make sure you're working on the develop branch for both the main repository and each submodule. If a submodule still shows errors, check the output of `git submodule status` to identify the issue.

- **Setting Up Your Local Environment:** If additional steps are needed to set up your local development environment (e.g., installing dependencies, setting up a local database, etc.), make sure to follow the guidelines in the repository's README or documentation.

This process ensures that:

- Each submodule is updated with the latest changes from its respective develop branch.
- The submodule reference in the main repository is correctly updated and committed.

Guide for Deploying and Configuring the WMS

For a detailed explanation of all configuration parameters, you should refer to the document: [CONFIGURATION PARAMETERS](#)

WMS Configuration and Activation Process

- Step 1: Verify that the Containers are Running

Assuming the repositories have been cloned into /develop, check the status of the containers by running the following commands:

- ✓ *Verify Database and Proxy Functionality*

```
cd /develop/wms/code/wms-common  
docker compose ps
```

Expected output (not all columns are shown):

```
SERVICE STATUS PORTS  
wms_db Up 3 days 0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp  
wms_npm Up 3 days 0.0.0.0:980->80/tcp, 0.0.0.0:981->81/tcp, 0.0.0.0:9443->443/tcp  
This verifies that the database is accessible on port 3306, and the proxy listens on 9443 for HTTPS traffic.
```

- ✓ *Verify WMS API Functionality*

```
cd /develop/wms/code/wms-core  
docker compose ps
```

Expected output:

```
SERVICE STATUS PORTS  
wms_core Up 3 days 0.0.0.0:5201->5000/tcp
```

This confirms that the API is accessible on port **5201**.

✓ *Verify WMS Admin Frontend Functionality*

```
cd /develop/wms/code/wms-admin
```

```
docker compose ps
```

Expected output:

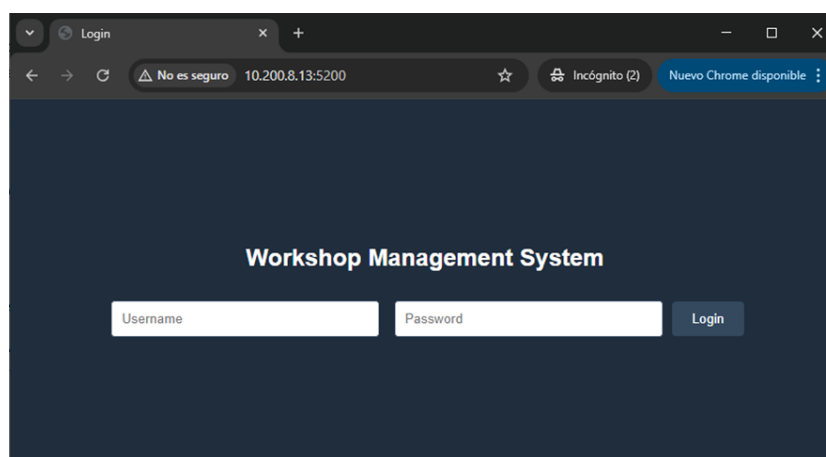
SERVICE	STATUS	PORTS
wms-admin	Up 3 days	0.0.0.0:5200->5000/tcp

This confirms that the admin frontend is accessible on port **5200**.

- Step 2: Verify Service Functionality

If the host server's IP address is **10.200.8.13**, test:

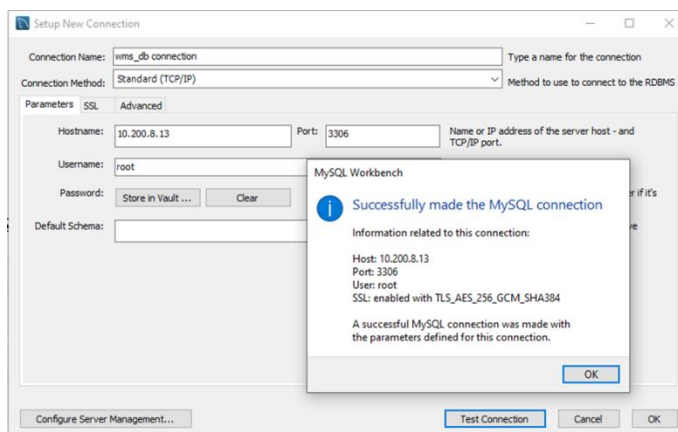
- **Admin Panel:** <http://10.200.8.13:5200>



API: http://10.200.8.13:5201



- **Database Connection:**
 - Username: root
 - Password: root_password (defined in docker-compose.yaml)



- Step 3: Verify Internet Traffic Routing to WMS

Traffic directed to public port **443** must be forwarded to the proxy (10.200.8.13:9443). To verify:

```
sudo tcpdump -i eth0 port 9443
```

(Replace eth0 with the correct network interface.)

- Step 4: Access the WMS Admin Panel

Go to <http://10.200.8.13:5200> and log in with:

- **Username:** admin
- **Password:** admin123

Since the WMS has not been configured yet, the connection will be rejected by the CMS, which will be indicated with a red icon. The message "**rejected**" will appear ("rejected: Connection rejected by CMS. An action is required in CMS admin console").

 **rejected: Connection rejected by CMS. An action is required in CMS admin console**

- Step 5: Configure the WMS

Each partner should use the following assigned values:

wms_uid	wms_hash	short_code	name
1	B7S4WWINCI8F69IIPREEJ62K9DSL986U	XAB	CFP Xabec
2	0CI5Z8YSQ3X5OP7M7WKWHHMF1MPCBBDM	FMP	FEMPA
3	FHWRTIS68F3Y94K6XV9DW17M818RTK5O	ENG	ENGIM
4	JPLM0YF2SFHWQOZJJS3Q7HL6WOEQ6FAW	KAD	KADKINE

1. Edit the configuration file:

```
nano /develop/wms/config.json
```

2. The following parameters will be adjusted according to the identifiers in the previous table and the domains resolved by the DNS servers pointing to the public IP of the center:



- **wms_uid** ⇒ According to the previous table
- **wms_api_url** ⇒ Based on the domain/DNS pointing to the public IP
- **auth_api_url** ⇒ Same as the previous one, but keeping /wms/any/sessions
- **wms_hash** ⇒ According to the previous table
- **name** ⇒ According to the previous table
- **short_code** ⇒ According to the previous table. Users registering will have the center's code followed by a number (FPA000001, FPA000002, FPA000003, ...)

3. Database connection must include the MySQL container's IP:

```
"db_host": "10.200.8.13:3306",
"db_name": "wms_db",
"db_user": "wms_user",
"db_password": "wms_password"
```

4. **Database Port:** Refer to the file wms/code/wms-common/docker-compose.yaml to identify the port where the database is accessible.

- The **IP address** should be the host machine running the Docker container.
- **Database Name, User, and Password:** Refer to the file wms/code/wms-common/inc/01_init_wms_db.sql to find the following parameters:
 - db_name
 - db_user
 - db_password

Ensure these parameters are consistent between the configuration file and the Docker/SQL initialization scripts.

4. Additional parameters. The other parameters will not be modified for now.

```
"register_frequency": "30",
"cms_api_url": "https://cms.vetworc.com",
"cms_api_key": "xxx",
"cms_public_api_key": "xxx",
"user_nodes_report_api_key": "secret_api_key",
"user_nodes_report_endpoint": "/wms/usernodes/report",
"keep_alive_sessions": "300",
"es_api_key": "cc",
"es_index": "user-units",
"es_url": "http://localhost:9200"
```

5. Save the changes and restart the service:

```
docker compose restart
```

- Step 6: Register with the CMS

1. Restart the WMS container:

```
docker compose restart wms-core
```

2. If the CMS does not recognize the MAC and HASH, "**rejected**" will appear in the admin panel
3. The CMS administrator must approve the connection and provide the **assigned HASH**.

- Step 7: Complete WMS Configuration

1. Edit the configuration file again:

```
nano /develop/wms/config.json'
```

2. Replace the wms_hash value with the assigned one.
3. Save the changes and restart the service:

```
docker compose restart
```

4. If configured correctly, a **green** success message will appear.

 **online: WMS is online**

If there is no network connection between WMS and CMS, the message will be "**offline: CMS seems not to be online or accessible**".

 **offline: CMS seems not to be online or accessible**

Important Notes

- **API Key Consistency:** Ensure the `wms_public_api_key` matches between `wms` and `wms-admin` configurations.
- **CMS Registration:** The WMS automatically registers with the CMS at intervals specified by `register_frequency` (in seconds). During this process, it sends its `wms_api_key` and information about resources to the CMS.
- **Database Alignment:** Verify that `db_host`, `db_name`, `db_user`, and `db_password` are aligned with the Docker container configuration and the database initialization script.
- **Monitoring Configuration:** Parameters related to Elasticsearch and user nodes do not require modifications unless monitoring services are being set up.

Comprehensive Guide to Container Creation Using Docker Compose

This guide provides a detailed step-by-step process for setting up the required Docker containers for the WMS (Workshop Management System). The setup uses three separate docker-compose.yml files, each corresponding to a different system component. It is crucial to start the containers in the correct order to ensure smooth operation.

Container Overview

Container	Service	Host Port	Comments
wms-common-wms_npm-1	Reverse Proxy	80, 81, 443	In production, this is the only container exposing ports.
wms-common-wms_db-1	Database (wms_db)	3306	Manages the WMS data-base.
wms-admin	WMS Management Frontend (wms-admin)	5200	Provides the admin interface for WMS.
wms-core	WMS API (wms-core)	5201	Hosts the WMS core API.

1. Setting Up wms-common

Configuration: docker-compose.yml

services:

vetworc_db:

image: 'mysql:8.0'

restart: always

environment:

MYSQL_ROOT_PASSWORD: 'root_password'





ports:

- '3306:3306'

security_opt:

- no-new-privileges:true

command: mysqld --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci

volumes:

- ./inc/01_init_wms_db.sql:/docker-entrypoint-initdb.d/01_init_wms_db.sql
- ./inc/02_populate_wms_db.sql:/docker-entrypoint-initdb.d/02_populate_wms_db.sql

networks:

- internal_net
- external_net

vetworc_npm:

image: 'jc21/nginx-proxy-manager:latest'

restart: always

ports:

- '80:80' # Puerto HTTP
- '81:81' # Interfaz de usuario web
- '443:443' # Puerto HTTPS

#user: "1001:1001" # Especifica un UID/GID que desees utilizar

security_opt:

- no-new-privileges:true

environment:

DB_MYSQL_HOST: "vetworc_db"

DB_MYSQL_PORT: 3306





DB_MYSQL_USER: "npm_user"

DB_MYSQL_PASSWORD: "npm_password"

DB_MYSQL_NAME: "npm_db"

depends_on:

- vetworc_db

volumes:

- npm_data:/data

- npm_letsencrypt:/etc/letsencrypt

networks:

- internal_net

- external_net

volumes:

npm_data:

npm_letsencrypt:

networks:

internal_net:

name: vetworc_internal_net

driver: bridge

internal: true

external_net:

name: vetworc_external_net

driver: bridge



Container Initialization

Before creating the image, it may be useful to modify the database initialization script (/develop/wms/code/wms-common/inc/01_init_wms_db.sql/01_init_wms_db.sql) to add the creation of a user with GRANT permissions.

```
-- Crear usuario fernando with password #myPassword

CREATE USER 'superuser'@'%' IDENTIFIED BY '#myPassword';

GRANT ALL PRIVILEGES ON *.* TO 'superuser'@'%' WITH GRANT OPTION;


cd /develop/wms/code/wms-common

sudo docker compose up -d
```

Checking Container Status

Once this is done, we can check the status of the containers:

```
docker compose ps
```

Expected output:

NAME	IMAGE	COMMAND	SERVICE	CREATED
wms-common-vetworc_db-1	mysql:8.0	"docker-entrypoint.s..."		
vetworc_db	19 seconds ago	Up 16 seconds	0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp	
wms-common-vetworc_npm-1	jc21/nginx-proxy-manager:latest	"/init"		
vetworc_npm	17 seconds ago	Up 14 seconds	0.0.0.0:80-81->80-81/tcp, :::80-81->80-81/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp	

We can view the logs of the database container:

```
docker logs --tail 1000 wms-common-wms_db-1

docker logs --tail 1000 wms-common-wms_npm-1
```

Connecting to the Database Server

We can connect to the database server and list the databases (the root password is specified in `docker_compose.yaml`):

```
docker exec -it wms-common-wms_db-1 sh
```

```
fernando@vetworc:/develop/wms/code/wms-common$ docker exec -it wms-common-wms_db-1 sh
```

```
sh-5.1# mysql -u root -p
```

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 10

Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
mysql> show schemas;
```

```
+-----+  
| Database      |  
+-----+  
| information_schema |  
| mysql          |  
| npm_db         |  
| performance_schema |
```



```
| sys |  
| wms_db |  
+-----+
```

6 rows in set (0.00 sec)

2. Setting Up wms-core

Configuration: docker-compose.yml

services:

wms_core:

container_name: wms_core

build: .

ports:

- "5200:5000"

volumes:

- ./app:/app

- ../system/config.json:/etc/wms/system/config.json

restart: always

networks:

- vetworc_internal_net

- vetworc_external_net

environment:

PORT: 5000

FLASK_DEBUG: 1

networks:

vetworc_internal_net:

external: true





```
vetworc_external_net:
```

```
  external: true
```

Container Initialization

```
cd /develop/wms/code/wms-core
```

```
sudo docker compose up -d
```

3. Setting Up wms-admin

Configuration: docker-compose.yml

services:

wms-admin:

```
  container_name: wms_admin
```

```
  build: .
```

```
  ports:
```

```
    - "5201:5000"
```

```
  volumes:
```

```
    - ./app:/app
```

```
    - ../../system/config.json:/etc/wms/system/config.json
```

```
  restart: always
```

```
  networks:
```

```
    - vetworc_internal_net
```

```
    - vetworc_external_net
```

networks:

```
vetworc_internal_net:
```

```
  external: true
```

```
vetworc_external_net:
```

```
  external: true
```



Container Initialization

```
cd /develop/wms/code/wms-admin
```

```
sudo docker compose up -d
```

4. Verify All Containers

```
sudo docker ps
```

Expected Output:

```
fc2802bf5308 wms-admin      0.0.0.0:5000->5000/tcp wms_admin
```

```
3550efdae40b wms-core      0.0.0.0:5030->5000/tcp wms_core
```

```
f3cac6bfe531 nginx-proxy-manager 0.0.0.0:80->80/tcp    wms-common-vetworc_npm-1
```

```
86e08c660430 mysql:8.0      0.0.0.0:3306->3306/tcp wms-common-vetworc_db-1
```

Production Notes

Replace localhost with the server's domain/IP.

Configure SSL using a reverse proxy (Nginx).

By following this guide, you can successfully deploy the WMS system using Docker Compose.



✓ Database Connection

mysql -u root -p

Password = root_password

Creating a development user with GRANT privileges:

CREATE USER 'user'@'%' IDENTIFIED BY 'password';

*GRANT ALL PRIVILEGES ON *.* TO 'user'@'%' WITH GRANT OPTION;*



✓ Nginx Proxy Manager Configuration Guide

Introduction

Nginx Proxy Manager (NPM) runs within one of the Docker containers in the WMS infrastructure. This container is the only one accessible from the host machine, as the other containers (such as the WMS API and WMS Admin) are configured to use an internal, isolated network that is not directly exposed to the host or the internet in a production environment. This setup ensures that only NPM handles the incoming traffic from the internet and securely proxies it to the internal services.

NPM acts as a reverse proxy, enabling communication between public internet URLs and the internal servers and services within our network. NPM ensures that requests coming from the internet are properly routed to the appropriate services within the internal network, such as the WMS API and the WMS Admin page.

NPM also manages SSL/TLS certificates for the services it proxies, ensuring that connections are secure and encrypted. By using NPM, we can easily configure and manage domain names, SSL certificates (using Let's Encrypt), and routing rules, all while maintaining a secure and organized architecture for accessing internal services.

Accessing Nginx Proxy Manager

Access the Nginx Proxy Manager Web Interface

- Ensure that your server is accessible over the network. Use the configured server IP to access the Nginx Proxy Manager web interface from your browser.

Access URL:

<http://10.200.8.13:81>

- The default user to log in to the Nginx Proxy Manager interface is:

Username: admin@example.com

Password: changeme

It's highly recommended to change the password as soon as possible.

Username: fernando.navarro@xabec.es

Password: #1212adsd

Ngix Proxy Manager Configuration for Subdomains

- Step 1: Login to Ngix Proxy Manager
 - ✓ Open the web browser and navigate to the URL <http://10.200.8.13:81>.
 - ✓ Enter the default login credentials (username and password) to access the NPM interface.
- Step 2: Configure Proxy Hosts

Once logged in to the Ngix Proxy Manager interface, follow these steps to configure the subdomains:

- ✓ Go to the Proxy Hosts section from the left-hand menu.
- ✓ Click the Add Proxy Host button to configure each subdomain.

Configuration for wms.xabec.vetworc.com (WMS API):

- Domain Names: wms.xabec.vetworc.com
- Scheme: http
- Forward Hostname / IP: 10.200.8.13
- Forward Port: 5030
- Block Common Exploits: Enabled (Recommended)
- SSL: Select the SSL tab.
- SSL Configuration:
 - SSL Certificate: Select Request a new SSL certificate.
 - Force SSL: Check this box to redirect HTTP to HTTPS automatically.
 - HTTP/2: Enable HTTP/2 for better performance.
 - Let's Encrypt: Enable Request a new SSL certificate from Let's Encrypt.

Email for Let's Encrypt: Enter your email to receive notifications about SSL certificate renewals.

DNS challenge: If port 80 is not accessible from the internet, you will need to use a DNS challenge to verify ownership of the domain. Enable the DNS challenge if required by your DNS provider.

- Click Save to add the proxy host.

Configuration for front.wms.xabec.vetworc.com (WMS Admin Page):

- Domain Names: front.wms.xabec.vetworc.com
- Scheme: http
- Forward Hostname / IP: 10.200.8.13
- Forward Port: 5000
- Block Common Exploits: Enabled (Recommended)
- SSL: Select the SSL tab.
- SSL Configuration:
 - SSL Certificate: Select Request a new SSL certificate.
 - Force SSL: Check this box to redirect HTTP to HTTPS automatically.
 - HTTP/2: Enable HTTP/2 for better performance.
 - Let's Encrypt: Enable Request a new SSL certificate from Let's Encrypt.

Email for Let's Encrypt: Enter your email to receive notifications about SSL certificate renewals.

DNS challenge: If port 80 is not accessible from the internet, you will need to use a DNS challenge to verify ownership of the domain. Enable the DNS challenge if required by your DNS provider.

- Click Save to add the proxy host.
- Step 3: Test the Configuration

Once the proxy hosts are set up, verify that the subdomains are properly configured:



- ✓ Test <http://wms.xabec.vetworc.com> and <http://front.wms.xabec.vetworc.com> to check if they correctly forward traffic to the respective services (WMS API and WMS Admin Page).
- ✓ Ensure that both subdomains are now accessible via HTTPS by visiting:

<https://wms.xabec.vetworc.com>

<https://front.wms.xabec.vetworc.com>

The browser should automatically redirect from HTTP to HTTPS, and the connection should be secure.

Troubleshooting

If the SSL certificate generation fails or the sites do not load, follow these steps:

- ✓ Check DNS Records: Ensure that the DNS records for both wms.xabec.vetworc.com and front.wms.xabec.vetworc.com point to the correct IP address of your server (10.200.8.13).
- ✓ Verify Ports: Make sure that ports 80 and 443 are open and accessible from the internet. If they are not, you may need to use the DNS challenge for Let's Encrypt.
- ✓ Check Logs: You can check the Nginx Proxy Manager logs for any errors related to the SSL certificate issuance or domain routing.

Conclusion

By following these steps, you have configured Nginx Proxy Manager to proxy traffic to the WMS API and WMS Admin Page, with SSL certificates provided by Let's Encrypt. Make sure to regularly check that your SSL certificates are renewed, and update any necessary DNS records if the IP address of your server changes.

This guide ensures that both the WMS API and the WMS Admin Page are accessible securely using HTTPS. If you encounter any issues, refer to the troubleshooting section or seek further assistance.



✓ Using GitHub to Manage Bugs and Feature Requests

This document outlines the process for using GitHub to report and manage bugs and feature requests during system testing. It is designed for users who will be testing the system and submitting feedback.

Reporting Bugs and Requesting Features

- Step 1: Accessing the GitHub Repository
 - ✓ Navigate to the GitHub repository for the project. Vetworc Issues
 - ✓ Ensure you are logged in with your GitHub account. If you do not have an account, you will need to create one.
- Step 2: Submitting a Bug Report
 - ✓ Go to the Issues tab of the repository.
 - ✓ Click on the green New Issue button.
 - ✓ Select the template labeled Bug Report (if available). This template will guide you through the information needed to report a bug.
 - ✓ Fill in the following details:
 - .1 Title: A short and descriptive title for the bug.
 - .2 Description: Include steps to reproduce the bug, the expected behavior, and what actually happened.
 - .3 Environment: Specify the environment where the bug was observed (e.g., browser, operating system, version number).
 - .4 Attachments: Include screenshots, logs, or any other supporting evidence if possible.

Important: It is crucial to provide all the necessary information to replicate the issue. Without sufficient details, it will not be possible to investigate or resolve the bug.

Click Submit new issue.

- Step 3: Requesting a New Feature
 - ✓ Go to the Issues tab of the repository.
 - ✓ Click on the green New Issue button.
 - ✓ Select the template labeled Feature Request (if available).
 - ✓ Fill in the following details:
 - .1 Title: A concise title describing the feature.
 - .2 Description: Provide a detailed explanation of the desired functionality, including why it is needed and any specific requirements.
 - .3 Use Case: Describe how this feature will be used.
 - .4 Attachments: Add mockups, diagrams, or other supporting documents if applicable.
 - ✓ Click Submit new issue.
- Step 4: Tracking the Status

Once you have submitted an issue, you can track its status by:

 - ✓ Returning to the Issues tab.
 - ✓ Searching for your issue using the search bar or filters (e.g., by labels, assignees, or status).
 - ✓ Monitoring updates and comments from the development team.

Configuration for GitHub Masters

As the GitHub repository master, you need to set up the repository to support this workflow.



- Step 1: Enable Issue Tracking
 - ✓ Go to the repository's Settings.
 - ✓ In the "Features" section, ensure that the Issues checkbox is checked.
- Step 2: Create Issue Templates
 - ✓ Go to the repository's root directory and create a folder named `.github/ISSUE_TEMPLATE`.
 - ✓ Add Markdown files for the issue templates, such as:

bug_report.md: For bug reports.

feature_request.md: For feature requests.

Example bug_report.md:

name: Bug Report

description: Report a bug in the system.

title: "[Bug]: "

labels: bug

assignees: "

Describe the Bug

A clear and concise description of the bug.

Steps to Reproduce

1. Go to '...'
2. Click on '...'
3. Scroll down to '...'



4. See error.

Expected Behavior

A clear and concise description of what you expected to happen.

Screenshots or Logs

If applicable, add screenshots or logs to help explain the problem.

Environment

- OS: [e.g., Windows 10, macOS 12.0]
- Browser: [e.g., Chrome, Firefox]
- Version: [e.g., 22.0]

Example feature_request.md:

name: Feature Request

description: Suggest a new feature for the system.

title: "[Feature Request]: "

labels: enhancement

assignees: "

Describe the Feature

A clear and concise description of the feature you would like to see.

Justification

Why is this feature important? How does it improve the system?

Use Case

Describe how this feature will be used.

Additional Context

Add any other context, screenshots, or ideas about the feature here.

- Step 3: Define Labels
 - ✓ Go to the Issues tab.
 - ✓ Click on Labels to manage labels for categorizing issues.
 - ✓ Add labels such as:
 - .1 bug: For bug reports.
 - .2 enhancement: For feature requests.
 - .3 high-priority: For critical issues.
 - .4 question: For general inquiries.
- Step 4: Assign Collaborators
 - ✓ Go to Settings > Collaborators and Teams.
 - ✓ Add team members who will triage and address the issues.
- Step 5: Configure Notifications

Encourage team members to subscribe to notifications for issue updates to ensure timely responses.

By following these steps, testers and developers can efficiently collaborate to identify and resolve issues, ensuring a smoother development process and higher quality system.



Contributing to the Vetworc Projectn

Table of Contents

- Introduction
- Setting Up the Development Environment
 - ✓ Cloning the Repository and Opening in VS Code
 - ✓ Installing Required Dependencies
 - ✓ Installing Visual Studio Code Extensions
 - ✓ Creating the Virtual Environment and Installing Dependencies
 - ✓ Creating the launch.json File
- Step-by-Step Guides
 - ✓ Cloning the Repository
 - ✓ Modifying a Document
 - ✓ Creating a New Document
 - ✓ Resolving Bugs or Developing Features
- Master Guide: Managing Pull Requests and Merges
 - ✓ Pull Request Review
 - ✓ Merging Develop into Main
- Reporting Bugs or Requesting Features
 - ✓ Bug Reporting
 - ✓ Feature Requests
- Branching Strategy and Best Practices

Introduction

Welcome to the Vetworc project! This guide provides detailed steps for contributing, managing branches, and ensuring your contributions align with the project's workflows and standards.



Setting Up the Development Environment

- Cloning the Repository and Opening in VS Code
 - ✓ Navigate to your development directory:

```
mkdir /develop && cd /develop
```

- Clone the repository:

```
git clone --branch develop --single-branch https://github.com/Vetworc/wms.git  
cd /develop/wms
```

- Initialize and update submodules:

```
git submodule update --init --recursive
```

- Open the repository in Visual Studio Code:

```
code .
```

Installing Required Dependencies

Before creating the Python virtual environment (venv) and installing the necessary dependencies, you need to make sure your system has the required packages. This ensures that when you create the virtual environment, you can compile and install the dependencies properly.

- **Update the system:**

```
sudo apt update
```

- **Install the required packages:**

```
sudo apt install build-essential python3-dev gcc
```

These packages are essential for compiling and installing Python dependencies, especially when certain packages need to be compiled during installation.

Note: Make sure Python is installed on the server before proceeding with these steps.

Installing Visual Studio Code Extensions

For an optimal development experience, we recommend installing the following Visual Studio Code extensions. Each extension plays a crucial role in enhancing your workflow when working with the Vetworc project.

- **Remote - SSH (Microsoft):**

Remote – SSH This extension allows you to connect to a remote server over SSH directly from Visual Studio Code. It's essential for working on remote servers where the code repository is hosted.

- **Docker (Microsoft):**

Docker - This extension provides integration with Docker, allowing you to manage containers, images, and networks directly within Visual Studio Code.

- **Git Graph (mhutchie):**

Git Graph visualizes your Git repository's commit history, making it easier to track changes and manage your branches.

- **Live Server (Ritwick Dey):**

This extension launches a local development server with live reload feature for static & dynamic pages. It's useful for quickly previewing changes to web pages.

- **Python (Microsoft):**

This extension provides Python language support in Visual Studio Code, including features like IntelliSense, debugging, and testing.

Creating the Virtual Environment and Installing Dependencies

After setting up Visual Studio Code and installing the necessary extensions, the next step is to create a Python virtual environment (venv) and install the required dependencies from the requirements.txt files.

- **Create a virtual environment:**

From the root directory of the cloned project, run the following command to create a Python virtual environment:

```
python3 -m venv venv
```

- **Activate the virtual environment:**



source venv/bin/activate

- **Install dependencies:**

Once the virtual environment is activated, the extension in Visual Studio Code will automatically detect the requirements.txt files and prompt you to install the dependencies. When the prompt appears, make sure to select both requirements.txt files if they are detected.

This step is essential to install the required packages for the project.

Important Warning: Do Not Work in the main Branch

Under no circumstances should you work directly in the main branch. All development, bug fixes, and feature additions must be done in the develop branch. This ensures that the main branch remains stable and ready for production releases. Always make sure to switch to the develop branch before starting any work:

- **Ensure you are on the develop branch:**

git checkout develop

git pull origin develop

- **If needed, create a new branch for your task:**

git checkout -b feature/<your-feature-name>

Creating the launch.json File

To debug specific components, configure the launch.json file as follows:

- **Stop relevant Docker containers:**

docker stop wms-admin wms-core

- **Create or edit the .vscode/launch.json file with the following content:**

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "name": "WMS-CORE",
```

```
"type": "python",
"request": "launch",
"program": "code/wms-core/app/app.py",
"console": "integratedTerminal",
"cwd": "/develop/wms"
},
{
"name": "WMS-ADMIN",
"type": "python",
"request": "launch",
"program": "code/wms-admin/app/app.py",
"console": "integratedTerminal",
"cwd": "/develop/wms"
}
]
```

- **Restart Docker containers after debugging:**

```
docker start wms-admin wms-core
```

Step-by-Step Guides

Cloning the Repository

Cloning the repository is necessary the first time you want to download the project to your local machine. It provides a local copy of the Vetworc code, including all files and submodules, allowing you to make changes, contribute, and work on new features or fixes. This step only needs to be done once unless you need to reset your local repository or clone it to a new environment.

- **Navigate to the desired directory:**

```
cd /develop
```

- **Clone the repository:**

```
git clone --branch develop --single-branch https://github.com/Vetworc/wms.git
```

- **Open the repository in Visual Studio Code:**

```
code .
```

- **Initialize and update submodules:**

```
git submodule update --init --recursive
```

Modifying a Document

Modifying a document is necessary when you need to improve or update an existing file, such as improving documentation, fixing errors, or adding more information to a document that is part of the project. It can also involve improving a piece of code or even developing new functionalities if the document you're editing pertains to such tasks. This is a common procedure for contributors who want to address bugs, enhance features, or improve clarity in the documentation.

- **Open the repository in VS Code:**

```
cd /develop/wms && code .
```

- **Edit the desired file.**

- **Stage and commit your changes:**

```
git add <file>
```

```
git commit -m "Update: Description of changes"
```

- **Push changes to a feature branch:**

```
git push origin feature/<feature-name>
```

- **Create a pull request on GitHub.**

Creating a New Document

Creating a new document is necessary when you need to add new information to the project, such as creating a new guide, tutorial, or specification document. It is also required when introducing a new type of content that is important to the project but was not previously included. This helps expand the project's documentation and functionality.

- **Navigate to the docs directory:**

```
cd /develop/docs
```

- **Create a new Markdown file and add content.**

- **Stage, commit, and push the file:**

```
git add <file>
```

```
git commit -m "Add: New document"
```

```
git push origin feature/<feature-name>
```

- **Open a pull request on GitHub.**

Resolving Bugs or Developing Features

When working to resolve bugs or develop new features, it is essential to create a new branch off of develop to ensure that the main branch remains stable while you make your changes. This process allows you to work on isolated changes, test them, and merge them into develop once they are ready. This is a routine task when adding new functionalities or fixing issues.

- **Ensure you are on the develop branch:**

```
git checkout develop
```

```
git pull origin develop
```

- **Create a new branch:**

```
git checkout -b fix/<bug-name> or feature/<feature-name>
```

- **Make changes, stage, commit, and push:**

```
git add .
```

```
git commit -m "Fix: Description of the fix or feature"
```

```
git push origin <branch-name>
```



- **Create a pull request to merge into develop.**

Master Guide: Managing Pull Requests and Merges

Pull Request Review

1. Verify that the PR includes only relevant changes.
2. Ensure submodules point to the correct develop branches.
3. Run unit and integration tests.
4. Approve and merge the PR into the develop branch.

Merging Develop into Main

1. Ensure all features and fixes in develop are production-ready.
2. Merge the develop branch into main:

git checkout main

git merge develop

3. Run final production tests and push:

git push origin main

Reporting Bugs or Requesting Features

Bug Reporting

1. **Go to the GitHub Issues page.**
2. **Submit a Bug Report:**
 - ✓ **Title:** Brief description of the bug.
 - ✓ **Details:** Steps to reproduce, expected vs. actual behavior, environment details.
 - ✓ **Attach screenshots or logs.**

Feature Requests

1. **Go to the GitHub Issues page.**
2. **Submit a Feature Request:**





- ✓ **Title: Brief description of the feature.**
- ✓ **Details: Functionality, use cases, and importance.**

Branching Strategy and Best Practices

- Use main for production-ready code.
- Use develop for feature development and bug fixes.
- Create feature or bug-fix branches from develop.
- Ensure meaningful commit messages and small, focused changes.

This document ensures a clear and organized approach for contributors to effectively collaborate on the Vetworc project.



Development and Branching Strategy

1. Introduction

This document outlines the branching strategy and workflow for the project that uses the main repository and its submodules. Two main branches have been created:

- **main:** This is the stable branch that contains production-ready code.
- **develop:** This is the development branch where new features are developed and bugs are fixed before being reviewed and merged into main.

Submodules will be used within the main repository. The submodules will also follow the same branching strategy and will have their own main and develop branches.

2. Branching Strategy

2.1. Main Repository

- **main branch:** This is the stable branch that should reflect the production code. Only the Master can perform merges into this branch after appropriate reviews.
- **develop branch:** This branch is where the development and bug fixes happen. All changes should be made to this branch and then reviewed by the Master before being promoted to main.

You can access the main repository (vetworc) at <https://github.com/Vetworc/vetworc.git>

2.2. Submodules

Each submodule has two main branches:

- **main branch:** Represents the stable version of the submodule.
- **develop branch:** Represents the development version for bug fixes and new features.

The submodules in this project include:

- common repository: main and develop branches
- wms-common repository: main and develop branches
- wms-core repository: main and develop branches
- wms-admin repository: main and develop branches
- docs repository: main and develop branches <https://github.com/Vetworc/docs.git>

2.3. Submodule Integration Process

It is important that submodules stay in sync with their respective develop branches. This means that before any changes are made to the main repository, the programmer must ensure that each submodule's reference points to the corresponding develop branch.

For more details consult REPOSITORIES

3. Developer Guide

3.2. Developing Features and Fixing Bugs

3.2.1. Create a Branch for Your Task

Each time you begin working on a new feature or bug fix, create a new branch from develop:

```
git checkout -b feature/new-feature
```

Or for bug fixes:

```
git checkout -b fix/bug-fix
```

3.2.2. Work on the Submodules

Ensure that all submodules are on their develop branches and make the necessary changes within them. Then, commit your changes in the submodules and return to the main repository. See REPOSITORIES for more details.

3.2.3. Commit and Push Changes

Commit your changes in both the main repository and the submodules:

```
git commit -m "Description of the change"
```

```
git push origin feature/new-feature
```

Also, remember to commit and push changes in the submodules after making modifications.

3.2.4. Create a Pull Request

Once you have finished working on a feature or bug fix, open a pull request (PR) to merge your changes into the develop branch of the main repository. Ensure that any submodule changes are included if necessary.

3.3. Bug Tracking and Issue Management

Before starting development or fixing bugs, create a corresponding issue in GitHub to track the progress of the task. Each issue should represent either a new feature or a bug that needs to be fixed.

- When creating a new issue for a bug, provide a clear description of the issue, including steps to reproduce and expected behavior.
- For new features, describe the feature, its functionality, and any relevant details.

Once the issue is addressed, link the PR to the issue so it can be tracked and closed once the PR is merged.

4. Conclusion

This strategy helps separate development and bug fixing from production-ready code. By managing the develop and main branches and following a workflow with submodules, collaboration is improved, and production stability is maintained.

Workflow Guidelines for Collaborators

Rules for Development

1. Do not merge directly into develop. Always create a Pull Request (PR) from your development branch.
2. Branch Naming Convention:
 - Use feature/<name> for new features.
 - Use bug/<name> for bug fixes.
 - Keep branch names descriptive, concise, and use lowercase with hyphens instead of spaces.

Table of Contents

1. Update the Local Develop Branch
2. Creating New Branches for Development
3. Developing and Committing Changes
4. Validation Testing
5. Discarding a Branch
6. Merging Changes into Develop
7. Notifying Collaborators of the Merge
8. Preventing Conflicts
9. Git Command Reference

Update the Local Develop Branch

Very Important: Always update the branch with the latest changes from the remote to avoid conflicts and ensure consistency:

1. Switch to the develop branch:
git checkout develop
2. Update the branch with the latest changes from the remote:
git pull origin develop

If you encounter conflicts or are unsure about your local state and prefer to completely replace your local version with the remote version:

1. Reset your local develop branch to match the remote:

```
git fetch origin
```

```
git reset --hard origin/develop
```

Creating New Branches for Development

To work on a new feature, create a branch following the convention:

```
feature/<branch-name>
```

Naming Guidelines for the Branch:

- Use descriptive and concise names.
- Use lowercase only and hyphens instead of spaces.
- Example: feature/new-functionality or feature/fix-login-bug.

Steps to create the branch:

1. Ensure you are on the base branch (such as develop):

```
git branch
```

```
git checkout develop
```

```
git pull
```

2. Create a new branch and switch to it:

```
git branch feature/<branch-name>
```

```
git checkout feature/<branch-name>
```

3. Or create and switch to the new branch directly in a single command:

```
git checkout -b feature/<branch-name>
```

Developing and Committing Changes

1. Make the necessary changes in your new branch.
2. Save your changes with the following commands:

```
git add .
```

git commit -m "Brief description of the changes made"

Validation Testing

- Perform unit or functional tests for the implemented changes.
- Use tools or testing environments based on the nature of the development.
- Ensure the code does not introduce errors or break existing functionality.

Discarding a Branch

If you decide that the changes made are not useful:

1 Return to the develop branch:

git checkout develop

2 Delete the local branch:

git branch -d feature/<branch-name>

3 If the branch is already on the remote, delete it as well:

git push origin --delete feature/<branch-name>

Merging Changes into Develop

Important: Do not merge changes directly into develop.

Instead:

1. Push your working branch to the remote repository:

git push origin feature/<branch-name>

2. Create a Pull Request (PR) in GitHub:

- Go to the GitHub repository.
- Click on the "Pull Requests" tab.
- Select "New Pull Request."
- Choose the base branch (develop) and compare it with your branch (feature/).
- Add a title and a description summarizing the changes.
- Submit the PR.

3. The repository owner will review and approve the PR before merging it into develop.

Notifying Collaborators of the Merge

Once the changes are merged into develop, collaborators should update their local develop branch:

1. Return to the develop branch:

```
git checkout develop
```

2. Pull the latest changes:

```
git pull origin develop
```

Preventing Conflicts

Conflicts can occur if:

- Multiple branches modify the same lines of code.
- Branches are based on different versions of develop.

To resolve conflicts:

- During a merge or pull, Git will indicate the files with conflicts.
- Edit the files to manually resolve the conflicts.
- Mark the conflicts as resolved:

```
git add <file>
```

- Continue with the merge or commit:

```
git commit
```

Recommendation: Keep your branch updated with develop regularly to minimize conflicts:

```
git merge develop
```

Git Command Reference



Category	Action	Command
Updating Branches	Switch to a branch	<code>git checkout <branch-name></code>
	Update the current branch	<code>git pull origin <branch-name></code>
	Reset branch to match the remote	<code>git reset --hard origin/<branch-name></code>
Branch Management	Create a branch without switching	<code>git branch <branch-name></code>
	Create and switch to a branch	<code>git checkout -b <branch-name></code>
	Delete a local branch	<code>git branch -d <branch-name></code>
	Delete a remote branch	<code>git push origin --delete <branch-name></code>
Working with Changes	Stage all changes	<code>git add .</code>
	Commit staged changes	<code>git commit -m "<message>"</code>
Merging and Conflict Resolution	Merge changes from another branch	<code>git merge <branch-name></code>
	Mark conflicts as resolved	<code>git add <file></code>
GitHub Pull Requests	Push a branch to the remote	<code>git push origin <branch-name></code>
GitHub status	Show current branch and status	<code>git status</code>

Bug Resolution and Feature Development Workflow

This document provides a structured approach to resolving bugs or developing new features in your project. It covers creating a dedicated branch, implementing changes, merging them, and performing cleanup both locally and on GitHub.

Step 1: Update the Local develop Branch

Ensure your local develop branch is up-to-date with the remote repository:

```
git checkout develop
```

```
git pull origin develop
```

Step 2: Create a Branch for Your Work

Create a new branch for the fix or feature. Use a descriptive name following the convention `bug/<issue-number>-<short-description>` or `feature/<short-description>`:

```
git checkout -b bug/5-disable-create-button
```

Step 3: Make Changes and Commit

- Implement the necessary changes in your branch.
- Stage the changes and create a descriptive commit message:

```
git add .
```

```
git commit -m "Fix: Disable create button [Closes #5]"
```

Use `[Closes #5]` in the commit message to automatically close the related issue upon merging.

Step 4: Push the Branch to Remote

Push the branch to the remote repository to make it available on GitHub:

```
git push origin bug/5-disable-create-button
```

4. GitHub Workflow

Step 5: Create a Pull Request (PR)

- ✓ Navigate to your repository on GitHub.
- ✓ Click "Compare & pull request" for the branch you just pushed.



- ✓ Set the base branch to develop.
- ✓ Provide a meaningful description if needed.
- ✓ Submit the PR.

Step 6: Merge the Pull Request

- ✓ Review the PR details. If there are no conflicts, GitHub will indicate that the branch can be merged automatically.
- ✓ Click "Merge pull request" and confirm.
- ✓ GitHub will automatically close the issue if [Closes #5] was included in the commit message.

5. Post-Merge Cleanup

Step 7: Update the Local develop Branch

After merging, ensure your local develop branch reflects the latest changes:

```
git checkout develop
```

```
git pull origin develop
```

Step 8: Delete the Feature Branch

Once the branch is merged and no longer needed:

Delete the branch locally:

```
git branch -d bug/5-disable-create-button
```

Delete the branch remotely:

```
git push origin --delete bug/5-disable-create-button
```

Summary of Commands

```
# Prepare the Environment
```

```
git checkout develop
```

```
git pull origin develop
```

```
git checkout -b bug/5-disable-create-button
```



Implement Fix

git add .

git commit -m "Fix: Disable create button [Closes #5]"

git push origin bug/5-disable-create-button

Update `develop` After Merge

git checkout develop

git pull origin develop

Cleanup

git branch -d bug/5-disable-create-button

git push origin --delete bug/5-disable-create-button

GitHub Actions

- Create Pull Request: Use the "Compare & pull request" button for the new branch.
- Review and Merge: Merge the PR into develop.
- Automatic Issue Closure: Ensure the commit message includes [Closes #<issue-number>] to close the issue automatically.

This workflow ensures a clean and organized process for resolving bugs or developing new features while maintaining the integrity of your repository.

WMS Service Configuration and Access Guide

1. Overview

Once the Workshop Management System (WMS) is operational, it is essential to install and configure the necessary services. When students connect to the WMS, they can select different cells or workshops to access. For example, the automation panel contains equipment enabling practical exercises such as PLC programming, motor control, and encoder management.

A student cannot use the cell's resources indefinitely when they are limited. Students receive connection time through the Time Voucher SCORM package. A teacher uses the WMS front end to create a SCORM package that grants a student a specific amount of connection time to a cell upon opening it.

Concept of Cell

A cell represents a group of devices that students can interact with. Student qualifications and progress are typically assessed at the cell level.

Examples of Cells

- a) PLC Networks: A collection of programmable logic controllers (PLCs) that students can program and interact with.
- b) Industrial Panels: A panel equipped with various devices such as:
 - A PLC
 - An electric motor
 - A frequency converter
 - An IP camera to monitor motor movement
 - A computer providing remote desktop services for programming and control

Cells are defined by specific parameters stored in the database table `wms_db.wms_cell`:

Parameter	Description
<code>short_code</code>	Short identifier name
<code>name</code>	Full name of the cell
<code>description</code>	Detailed description

Adding Cells to the WMS

Once you determine the values for `short_code`, `name`, and `description`, use the WMS administration interface (`wms-admin`) to register the new cell in the system.

Service Table Example:

Workshop management

Cells

Services

User units

Cells Management

Create Cell

Code	Name	Status	Actions		
96UG	Célula Siemens.	online	Delete	Edit	Details
F0G0	Célula 2	offline	Delete	Edit	Details
4TFD	Célula 32	offline	Delete	Edit	Details
VVHL6	Celula 4	online	Delete	Edit	Details
RQ1SI	2342	offline	Delete	Edit	Details
VPOYF	Panel del taller 2	online	Delete	Edit	Details
NW9BI	Un célula más	offline	Delete	Edit	Details
YROSN	Robot	online	Delete	Edit	Details
D20IB	Nueva célula	online	Delete	Edit	Details
VZ0TT	Panel VR	online	Delete	Edit	Details
W15FC	Automation panel	online	Delete	Edit	Details

Concept of service

When accessing the BMS (Building Management System) to connect to a cell, the BMS displays the services available for that cell. These services are defined and published by the WMS, which communicates the details to the BMS.

Examples of Services

- A Wiki Media page
- Remote desktop service
- IP camera streaming

Services are defined by specific parameters stored in the database table `wms_db.wms_service`:

Parameter	Description
name	Name of the service
type	Specifies the service type. This determines how the BMS uses the endpoint to display the service. Current types: <code>web</code> , <code>rdp</code> , <code>streaming</code> .
endpoint	The URL the BMS should use to access the service. For example, for a <code>web</code> service, this would be the URL embedded in an <code>iframe</code> .
config	Additional configuration parameters (reserved for future use)

Service Types

Service Type	Description	Prerequisites
rdp	Remote Desktop	Requires the RDP repository
streaming	Access to IP cameras	Requires the Streaming repository
wikimedia	Wiki Media service	-

Service Table Example:

Workshop management

[Cells](#)
[Services](#)
[User units](#)

Services Management

Create Service

Cell ID	Cell Name	Service Name	Type	Consumes time	Qualified	Status	Actions	
6	Célula Siemens.	guacamole	rdp	No	No	online	Delete	Edit
6	Célula Siemens.	omron	doc	No	No	online	Delete	Edit
6	Célula Siemens.	xabec	doc	No	No	offline	Delete	Edit
17	Panel del taller 2	Cámara IP	stream	No	No	offline	Delete	Edit
17	Panel del taller 2	Cámara IP 2	stream	No	No	offline	Delete	Edit
18	Un célula más	Servicio nuevo	doc	No	No	online	Delete	Edit
19	Robot	Control remoto	rdp	No	No	online	Delete	Edit
22	Panel VR	Gafas 3D	rdp	No	No	online	Delete	Edit
23	Panel del taller 1	Portátil	rdp	No	No	online	Delete	Edit
23	Panel del taller 1	Info taller	doc	No	No	online	Delete	Edit

© 2025 Services Manager

3. Services Provided by the Automation Panel

When a student wants to perform automation tasks, they need to use TIA Portal to program the PLC and other devices in the panel. Additionally, they need to access documentation about the equipment (e.g., IP addresses, manuals, etc.). For this reason, the following services are provided:

Service	Comments
Remote desktop service	Allows access to a computer with the necessary software (TIA Portal)

Service	Comments
Documentation	Documentation, manuals, user guides, technical references, etc.
Learning resources	Various educational resources that the student can use
Live video	Live video streaming that allows viewing the equipment and its movements

Service 1. Remote Desktop Service

This service provides students with remote access to a wall-mounted computer to program PLCs and variable frequency drives (VFDs) using the TIA Portal software.

- Access Requirements: Only qualified students with proven competence can use this service.
- Infrastructure Needed:
 - A server running Apache Guacamole (Installation Guide)
 - One or more computers running TightVNC to provide remote desktops (User Node Repository)
- Service URL: <https://panel.automation.xabec.es/guacamole>
- Time consuming: Yes.

Service 2. Documentation and Information Portal (Wikimedia Service)

This service offers a web-based documentation platform (Wikimedia) where students can consult information, manuals, and guides regarding the equipment and devices of the panel.

- Access Requirements: Requires qualification due to sensitive information (IP addresses, configurations, etc.).



- Infrastructure Needed:
 - A server running wikimedia (Installation Guide)
- Service URL: <https://doc.automation.xabec.es>
- Time consuming: No.

Service 3. Educational Resources Repository

Students can access and download publicly available educational materials.

- Access Requirements: Open to all users.
- Infrastructure Needed:
 - A server running a webpage.
- Service URL: <https://resources.automation.xabec.es>
- Time consuming: No.

Service 4. Live Video Streaming

An IP camera provides a real-time video stream, allowing students to observe the physical operations of the panel or cell. The video stream should be accessible via a standard web browser (Chrome, Firefox, etc.).

- Access Requirements: Only available to qualified students.
- Infrastructure Needed:
 - A video streaming server.
- Service URL: <https://video.automation.xabec.es>
- Time consuming: Yes.

4. Service Summary Table

Service Name	URL (endpoint)	Cell	Access Type	Consumes Time
Computer	panel.automation.xabec.es/guacamole	Automation Panel	Qualified Only	Yes
Documentation	doc.automation.xabec.es	Automation Panel	Qualified Only	No
Resources	resources.automation.xabec.es	Automation Panel	Public	No
Live Video	video.automation.xabec.es	Automation Panel	Qualified Only	Yes

5. Time Consumption Policy

Understanding which services consume connection time is crucial for efficient resource usage:

- **Non-Time-Consuming Services:** Accessing documentation and resources (even if restricted to qualified users) does not consume allocated connection time.
- **Time-Consuming Services:** Limited resources, such as the remote desktop and live video streaming, do consume connection time. These services are considered scarce and are subject to usage limits to ensure fair access for all students.

6. Service Parameters

- Each service is characterized by the following parameters:



- **Service Name:** Identifies the service.
- **URL (Endpoint):** The address students must visit to access the service. No es una dirección que utiliza el alumno. Se trata de una dirección interna que el sistema utiliza para redireccionar al alumno.
- **Providing Cell:** The specific cell or workshop offering the service.
- **Access Type:** Indicates whether the service is public or requires qualification.
- **Time Consumption:** Specifies whether using the service consumes connection time.

7. Example Service Access Flow

When a student accesses the WMS:

- They select a cell (e.g., the automation panel).
- The system displays available services based on the student's qualification and connection time balance.
- Public services (e.g., educational resources) are always accessible.
- Restricted services (e.g., RDP or video streaming) are available only if the student:
 - Has sufficient connection time (granted via the Time Voucher SCORM package).
 - Meets the required qualifications.

If the student lacks the necessary time or qualifications, the system will notify them with appropriate guidance.

8. Installation of the RDP Guacamole Service

To provide the "RDP Guacamole Service", you must set up:

- A Guacamole server (refer to the dedicated installation guide).
- User nodes (RDP desktops) with TightVNC to enable remote desktop connections (Vetworc User Node Repository).

The service should be accessible at: <https://panel.automation.xabec.es/guacamole>

Guide to Registering a Service in the WMS System

The WMS management system allows administrators to register and configure services that students can access based on specific rules and restrictions. This guide explains the process, concepts, and steps required to successfully add a new service.

1. Understanding Key Concepts

Before adding a service, it is important to understand how access is managed:

Service Access Rules

- Voucher Requirement (Requires Voucher)
 - Some services require a time voucher for students to access them.
 - A time voucher is a SCORM package available in the LMS.
 - When a student opens the SCORM package, the system registers the authorized workshop cell and the allowed time period.
 - After this, the student can make reservations to connect to that cell.
 - The teacher does not grant access directly but provides the time voucher through the LMS.
- Qualification Requirement (Requires Qualification)
 - Some services require the student to pass a qualification test before access is granted.
 - If enabled, only qualified students will be able to access the service.
- Free Access Services
 - If Voucher Requirement and Qualification Requirement are both set to No, the service will be open to all Vetworc users with no restrictions.

Service Endpoint

All services are web-based and require an endpoint (URL) for students to access them.

Example for an RDP service:

<https://rdp.automation.xabec.es/guacamole>

This URL leads to Xabec's Guacamole remote desktop server.

2. Checklist Before Registering a New Service

Before proceeding, ensure you have the following information:

Parameter	Description	Value
Cell	The workshop cell offering this service.	
Service Name	The name displayed to students.	
Service Type	RDP (Remote Desktop), Doc (Documentation), or Stream (Streaming).	
Status	Online (available if reserved) or Offline (completely disabled).	
Requires Voucher	Yes (students need a SCORM-based time voucher) or No (open access).	
Requires Qualification	Yes (students need qualification) or No (open access).	
Endpoint	The URL where the service is hosted.	
Config JSON	Reserved for future configuration options.	

3. Steps to Register a New Service

- Step 1: Access the WMS Management System

Log in as an administrator.

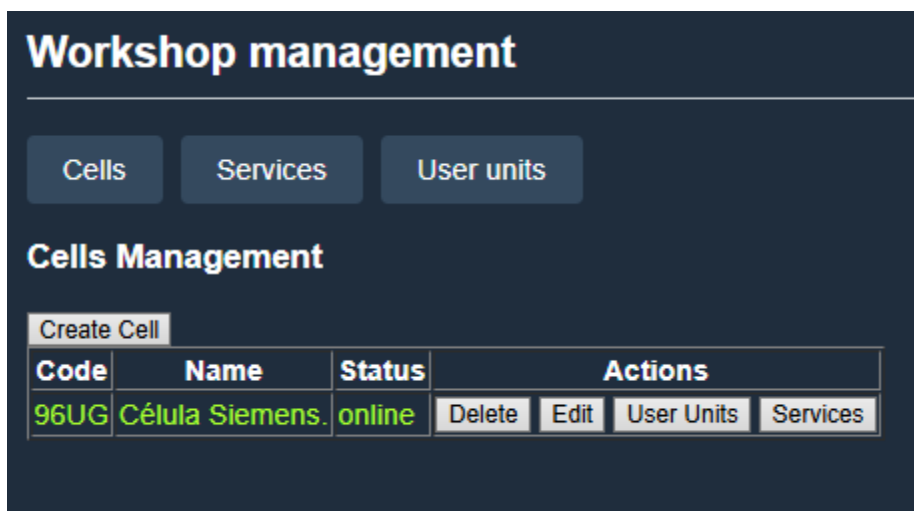


<https://front.wms.xabec.vetworc.com>

Username: admin

Password: admin123

Navigate to Workshop Management > Cells.



The screenshot shows the 'Workshop management' interface. At the top, there are three buttons: 'Cells', 'Services', and 'User units'. Below these, the 'Cells Management' section is visible. It includes a 'Create Cell' button and a table with the following data:

Code	Name	Status	Actions			
96UG	Célula Siemens.	online	Delete	Edit	User Units	Services

Check for the Automation panel. If not present, create it.

Create Cell

Name

Automation Panel

Description

Automation panel in workshop 1

Status

Online

Save

Cancel

Workshop management

Cells

Services

User units

Cells Management

Create Cell

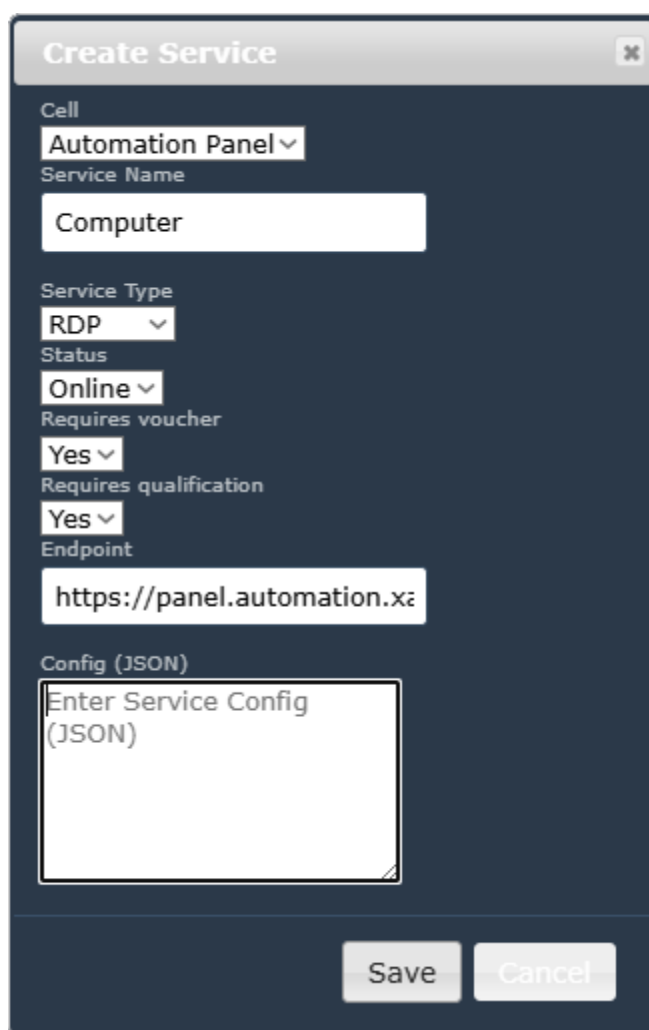
Code	Name	Status	Actions			
96UG	Célula Siemens.	online	Delete	Edit	User Units	Services
DTR1L	Automation Panel	online	Delete	Edit	User Units	Services

Step 2: Open the "Services" Section

Click on "Services" to view a list of existing services.

Under Service tab, you can edit existing services or add new ones.

- Cell: Automation panel
- Name: Computer
- Service type: RDP
- Status: online
- Requires voucher: Yes
- Requires qualification: Yes
- Endpoint: <https://rdp.automation.xabec.es/guacamole>



Create Service

Cell
Automation Panel ▾

Service Name
Computer

Service Type
RDP ▾

Status
Online ▾

Requires voucher
Yes ▾

Requires qualification
Yes ▾

Endpoint
<https://panel.automation.xabec.es/guacamole>

Config (JSON)
Enter Service Config (JSON)

Save Cancel

Create all services and verify they appear in the list:

Workshop management

Cells Services User units

Services Management

Create Service

Cell ID	Cell Name	Service Name	Type	Consumes time	Qualified	Status	Actions	
6	Célula Siemens.	guacamole	rdp	No	Yes	online	Delete	Edit
6	Célula Siemens.	omron	doc	No	No	online	Delete	Edit
6	Célula Siemens.	xabec	doc	No	No	offline	Delete	Edit
25	Automation Panel	Computer	rdp	Yes	Yes	online	Delete	Edit
25	Automation Panel	Documentation	doc	No	Yes	online	Delete	Edit
25	Automation Panel	Resources	doc	No	No	online	Delete	Edit
25	Automation Panel	Live Video	stream	Yes	Yes	online	Delete	Edit

Service Table Example:

Workshop management

Cells Services User units

Services Management

Create Service

Cell ID	Cell Name	Service Name	Type	Consumes time	Qualified	Status	Actions	
6	Célula Siemens.	guacamole	rdp	No	No	online	Delete	Edit
6	Célula Siemens.	omron	doc	No	No	online	Delete	Edit
6	Célula Siemens.	xabec	doc	No	No	offline	Delete	Edit
17	Panel del taller 2	Cámara IP	stream	No	No	offline	Delete	Edit
17	Panel del taller 2	Cámara IP 2	stream	No	No	offline	Delete	Edit
18	Un célula más	Servicio nuevo	doc	No	No	online	Delete	Edit
19	Robot	Control remoto	rdp	No	No	online	Delete	Edit
22	Panel VR	Gafas 3D	rdp	No	No	online	Delete	Edit
23	Panel del taller 1	Portátil	rdp	No	No	online	Delete	Edit
23	Panel del taller 1	Info taller	doc	No	No	online	Delete	Edit

© 2025 Services Manager

Step 3: Create a New Service

1. Click "Create New Service".
2. Fill in the fields using the checklist information.
3. Ensure the status is set correctly:
 - Online → Available for authorized students.
 - Offline → No connections allowed.
4. Configure access rules:
 - If Requires Voucher = Yes, students must obtain a SCORM time voucher.
 - If Requires Qualification = Yes, only qualified students will have access.
 - If both are No, the service will be open to all Vetworc users.
5. Enter the service endpoint URL (e.g., <https://rdp.automation.xabec.es/guacamole>).
6. (Optional) Enter Config JSON (not currently in use).

Step 4: Save and Verify

1. Click Save to register the new service.
2. Verify that the service appears in the list of services.
3. Ensure it is set to Online if it should be available immediately.

4. Conclusion

Once registered, the service will be available to students based on the configured access rules. If any changes are needed, the service settings can be modified from the Workshop Management > Services section.

This guide provides a clear and structured process to ensure that services are properly registered and managed in the WMS system.

Documentation for Specific Services

For detailed instructions on installing and configuring each service, refer to the following documents:





1. RDP SERVICE
2. WEB SERVICE
3. WIKI SERVICE
4. STREAMING SERVICE

Key Improvements and Additional Details

1. Terminology Clarifications: Terms like "cell" and "service" are defined more clearly for easier understanding.
2. Detailed Examples: Concrete examples of cells and services make the guide more actionable.
3. References to Additional Resources: Links to service-specific documentation provide a comprehensive learning path.
4. Database Integration: Clear mapping of parameters to database tables ensures ease of implementation.

This guide provides a structured approach to setting up and managing cells and services within the WMS ecosystem.

New Service



WMS Quick Installation Guide

1. Prerequisites

- Ubuntu server with Docker and Docker Compose installed.
 - SERVER INSTALLATION
 - INSTALLING DOCKER
- SSH access to the server.
- Internet access from the server.
- GitHub user and token to clone repositories.
 - GITHUB USER TOKEN

Refer to the PREREQUISITES QUICK INSTALLATION GUIDE for a quick reference on prerequisite installation.

2. Accessing the Server

Connect to the server via SSH:

```
ssh fernando@vetworc
```

Expected output:

```
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-52-generic x86_64)
```

```
...
```

```
Last login: Sun Feb 23 09:54:13 2025 from 192.168.1.31
```

3. Preparing the Working Environment

Create the working directory and assign permissions:

```
sudo mkdir -p /develop
```

```
sudo chown -R fernando:fernando /develop
```

```
cd /develop
```

4. Cloning Repositories

Clone the WMS repository:

```
git clone --branch develop --single-branch https://github.com/Vetworc/wms.git
```

Expected output:

```
Cloning into 'wms'...
```

```
Receiving objects: 100% (619/619), 49.63 MiB | 15.13 MiB/s, done.
```

```
Resolving deltas: 100% (305/305), done.
```

Clone the documentation:

```
git clone --branch develop --single-branch https://github.com/Vetworc/docs.git
```

Expected output:

```
Cloning into 'docs'...
```

```
Receiving objects: 100% (301/301), 1.50 MiB | 8.35 MiB/s, done.
```

5. Configuration

At this development point is only recommended to modify the following parameters (minimum required). Other will be modified in the future as the security elements become tested.

Edit the configuration file:

```
nano /develop/wms/config.json
```

Modify the following parameters:

- wms-admin:
 - wms_uid
 - wms_api_url
 - auth_api_url
- wms:



- wms_uid
- wms_hash
- name
- short_code
- wms_api_url
- db_host

Note: The wms_public_api_key value must be the same in both wms-admin and wms sections.

6. Launching Containers

6.1. Common Containers

```
cd /develop/wms/code/wms-common
```

```
sudo docker compose up -d
```

Check the containers:

```
docker compose ps
```

Expected output (key columns only):

NAME	SERVICE	STATUS	PORTS
wms-common-wms_db-1	wms_db	Up About a minute	0.0.0.0:3306->3306/tcp
wms-common-wms_npm-1	wms_npm	Up About a minute	0.0.0.0:980->80/tcp

6.2. Verifying the Database

```
docker exec -it wms-common-wms_db-1 sh
```

```
mysql -u root -p
```

Enter root password as defined in docker-compose.yaml



List databases:

```
show schemas;
```

Expected output:

```
+-----+
| Database      |
+-----+
| information_schema |
| mysql         |
| npm_db        |
| wms_db        |
+-----+
```

6.3. Starting wms-core

```
cd /develop/wms/code/wms-core
```

```
sudo docker compose up -d
```

Check its status:

```
docker ps
```

Expected output:

NAME	SERVICE	STATUS	PORTS
wms_core	wms_core	Up About a minute	0.0.0.0:5201->5000/tcp

6.4. Starting wms-admin

```
cd /develop/wms/code/wms-admin
```

```
sudo docker compose up -d
```

Check again:

```
docker ps
```

Expected output:

NAME	SERVICE	STATUS	PORTS
wms_admin	wms_admin	Up About a minute	0.0.0.0:5200->5000/tcp

7. Final Verification

Access the frontend using the server's IP address and the corresponding port (e.g., 5200):

http://<server_ip>:5200

Note: Final verification should be done using the server's IP. Access via URL requires reverse proxy configuration, which will be covered in the "07_NPM.md" guide.

If everything is properly configured, you should be able to see the WMS user interface.

Notes

Ensure that all API keys and URLs are correctly configured.

The database must be fully initialized to avoid connection errors.

Port numbers may vary depending on your configuration.

Server prerequisites quick installation guide

Here is the step-by-step guide to install Git, Docker, and Docker Compose on Ubuntu 22.04:

1. Install Git

```
sudo apt update
```

```
sudo apt install git -y
```

2. Install Docker

```
sudo apt update
```

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o  
/usr/share/keyrings/docker-archive-keyring.gpg
```

```
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt update
```

```
sudo apt install docker-ce docker-ce-cli containerd.io -y
```

3. Verify Docker

```
sudo systemctl start docker
```

```
sudo systemctl enable docker
```

```
sudo docker --version
```

4. Install Docker Compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

5. Verify Docker Compose

```
docker-compose --version
```

With these steps, you will have Git, Docker, and Docker Compose installed on your Ubuntu 22.04 server.

Using a GitHub user and token

Here is a minimal step-by-step tutorial for using a GitHub user and token (classic) to clone repositories:

1. Create a GitHub Account (if you don't have one)

- Go to GitHub.
- Sign up for a new account or log in if you already have one.

2. Generate a Personal Access Token (Classic)

- Go to GitHub Tokens Settings.
- Click on Generate new token.
- Give your token a descriptive name (e.g., "Access token for cloning repositories").
- Under Select scopes, check the box for repo (this will allow you to clone private repositories).
- Optionally, you can select additional scopes depending on your needs.
- Click Generate token.
- Copy your newly generated token and store it securely (you will not be able to see it again).

3. Install Git

If you don't have Git installed, follow these commands:

```
sudo apt update
```

```
sudo apt install git -y
```

4. Clone a Repository Using Your GitHub Token

To clone a repository, use the following command:

```
git clone  
https://<YOUR_GITHUB_USERNAME>:<YOUR_TOKEN>@github.com/<USERNAME>/<REPOSITORY_NAME>.git
```

- Replace <YOUR_GITHUB_USERNAME> with your GitHub username.
- Replace <YOUR_TOKEN> with the personal access token you generated.



- Replace <USERNAME> and <REPOSITORY_NAME> with the owner and name of the repository you want to clone.

For example:

```
git clone  
https://yourusername:yourgeneratedtoken@github.com/username/repository.git
```

5. Verify the Clone

After running the command, Git will clone the repository to your local machine. Navigate to the directory where it was cloned:

```
cd <REPOSITORY_NAME>
```

You should now have a local copy of the GitHub repository.

With these steps, you can use your GitHub username and personal access token to clone repositories, including private ones.



Configuration reference guide for config.json

This file defines all the configuration parameters for the WMS system, including the front-end (wms-admin) and the back-end API (wms). It is structured in two main sections:

- wms-admin: Configuration for the WMS front-end interface.
- wms: Configuration for the internal backend, database, and CMS/API integration.

Section: wms-admin — Front-End Configuration

This section defines the parameters used by the WMS front-end interface (wms-admin). It includes identifiers, API URLs and keys, authentication settings, SCORM integration, and optional metrics reporting from user nodes.

a) WMS Identification

Parameter	Example Value	Explanation
wms_uid	"1"	Unique identifier of the WMS. Must match the value in the wms section.

b) WMS API Access

Parameter	Example Value	Explanation
wms_api_url	"https://wms.xabec.vetworc.com"	URL to access the WMS API.
wms_public_api_key	"api_key for wms-admin"	API key used by the front-end. Must match the same key in the wms section.

c) User Authentication

Parameter	Example Value	Explanation
auth_api_url	"https://wms.xabec.vetworc.com/wms/any/sessions"	URL used for authenticating users. Should point to the WMS API unless a custom one is used.
auth_api_key	"api_key for wms-admin"	API key for user authentication. Usually the same as wms_public_api_key.

d) User Node Reporting (Not used in version 1)

Parameter	Example Value	Explanation
user_nodes_report_api_key	"mi_api_key_secret para usernodes"	API key for remote desktops reporting usage data.
user_nodes_report_endpoint	"/wms/usernodes/report"	Relative endpoint for reporting metrics.

e) SCORM Package Configuration

Parameter	Example Value	Explanation
scorm_api_url	"https://front.wms.xabec.vetworc.com"	URL used by SCORM packages to communicate with the WMS.
scorm_api_key	"api_key used by scorm packages to access wms-admin endpoints"	API key embedded in SCORM packages. Changing this key breaks existing packages.

Section: wms — Backend & API Configuration

This section defines the internal settings for the WMS back-end, including database credentials, CMS integration, connection control, and future support for dashboards and remote desktops.

a) WMS Identification

Parameter	Example Value	Explanation
wms_uid	"1"	Unique identifier of the WMS instance.
wms_hash	"B7S4WWINCI8F69IIPREEJ62K9DSL986U"	Internal unique hash.
name	"Xabec"	Display name of the WMS.
short_code	"XAB"	Short code used to identify this instance.

b) WMS API Access

Parameter	Example Value	Explanation
wms_api_url	"https://wms.xabec.vetworc.com"	URL to the internal WMS API.
wms_api_key	"api_key used by cms to access wms endpoints"	Used by the CMS. Automatically provided during registration.
wms_public_api_key	"api_key for wms-admin"	Used by the front-end. Should match the key in the wms-admin section.

c) CMS Communication

Parameter	Example Value	Explanation
cms_api_url	"https://cms.vetworc.com"	URL of the CMS API.
register_frequency	"30"	How often (in seconds) the WMS reports and renews its credentials with the CMS.
cms_registration_api_key	"api_key for cms registration"	API key used to authenticate with the CMS. Must match CMS config.

d) Database Connection

Parameter	Example Value	Explanation
db_host	"192.168.1.20:3306"	Database host and port. Usually localhost or internal IP.
db_name	"wms_db"	Name of the WMS database.
db_user	"wms_user"	Username used to access the DB.
db_password	"wms_password"	Password for DB user.

e) Session Control

Parameter	Example Value	Explanation
cell_connection_keep_alive_timeout	300	Idle timeout in seconds before the WMS closes the student's session.

f) Dashboard Configuration (Not used in version 1)

Parameter	Example Value	Explanation
es_api_key	"cc"	API key for Elasticsearch (future use).
es_index	"user-units"	Elasticsearch index name.
es_url	"http://localhost:9200"	Elasticsearch server URL.

g) User Node Configuration (Not used in version 1)

Parameter	Example Value	Explanation
user_nodes_report_endpoint	"https://wms.xabec.vetworc.com/wms/uunits/report"	Full URL for metrics reporting.
user_nodes_report_api_key	"mi_api_key_secretaria_para_usernodes"	API key for remote desktops.
user_nodes_keepalive	"300"	Interval in seconds between keep-alive messages.

Full Example Configuration

```
{
  "wms-admin": {
    "wms_uid": "1",
    "wms_api_url": "https://wms.xabec.vetworc.com",
    "wms_public_api_key": "api_key for wms-admin",
    "auth_api_url": "https://wms.xabec.vetworc.com/wms/any/sessions",
    "auth_api_key": "api_key for wms-admin",
    "user_nodes_report_api_key": "mi_api_key_secretaria_para_usernodes",
```

```
"user_nodes_report_endpoint": "/wms/usernodes/report",
"scorm_api_url": "https://front.wms.xabec.vetworc.com",
"scorm_api_key": "api_key used by scorm packages to access wms-admin end-
points"
},
"wms": {
  "wms_uid": "1",
  "wms_hash": "B7S4WWINCI8F69IIPREEJ62K9DSL986U",
  "name": "Xabec",
  "short_code": "XAB",
  "wms_api_url": "https://wms.xabec.vetworc.com",
  "wms_api_key": "api_key used by cms to access wms endpoints",
  "wms_public_api_key": "api_key for wms-admin",
  "cms_api_url": "https://cms.vetworc.com",
  "register_frequency": "30",
  "cms_registration_api_key": "api_key for cms registration",
  "db_host": "192.168.1.20:3306",
  "db_name": "wms_db",
  "db_user": "wms_user",
  "db_password": "wms_password",
  "cell_connection_keep_alive_timeout": 300,
  "es_api_key": "cc",
  "es_index": "user-units",
  "es_url": "http://localhost:9200",
  "user_nodes_report_endpoint":
  "https://wms.xabec.vetworc.com/wms/uunits/report",
```



```
"user_nodes_report_api_key": "mi_api_key_secreta_para_usernodes",  
"user_nodes_keepalive": "300"  
}  
}
```

